Basic Python Programming

Roudranil Das

PDSP Aug-Nov 2022
Roll No: MDS202227
roudranil@cmi.ac.in

September 7, 2022

Contents

1 Introduction

So basic question is what is a programing language? Basically, writing **systematic procedure** in **precise notation**.

The systematic procedure is the algorthim and the precise notation is the programming language. Nature of instruction varies according to who is doing the job. Comparing this with an example of workers, we can describe it as following:

1. outsourcing → light instructions, leave to them to figure out
2. experienced → high level checklist
3. unfamiliar → very explicit checklist

Here we use python, very HLL.

2 August 16, 2022

2.1 Values and types

Note that, division always produces a float. Exponention is an int if both exponent and base is int, otherwise is a float. Integer division returns integer only if both arguments are int. Otherwise it will be float.

- a numberic value of '0' is `False`
- an emoty lsit is `False`
- an empty string is `False`
- anything that is not `False` is `True`

```
y = 5
7 and y, not(7 and y)
# [out] (5, False)

```

There is also no serious limit on size of integers. For example:

```
x = 2**84 # wont fit in 64 bits
y = 3**91
x * y
# [out] 506470104485618930545274704870272602283938237278635903874283341873152

```

Data type determines what operations are allowed. For example `len(x)` does not make sense if value of x is not an iterable. Names inherit their type from the values they currently hold. Thus it is not a good practice to use the same name for different types of values.

Control flow is a sequence of statements controlling the flow of execution, by using conditional statements, assignment statements, iteration statements.

2.2 Functions

Templates for re-usable code. Instantiate with different arguments.

3 August 23, 2022

3.1 Something to remember about list indexing

`list[x:y]` will work when $x \leq y$. If any one of them is negative, then it will be interpreted as the corressponding negative index ($length - x$). However if even that is beyond the length of the list, then it defaults to one of the following cases: if lhs is beyond length then to 0, if rhs is beyond the length then to length-1.

`list[:]` will return a **copy** and **not** a **slice** of the list.

3.2 Mutable and Immutable values

```
x = 7
y = x
x = x + 1
(x,y)
# (8,7)
```

```
l1 = [1,2,3]
l2 = l1 # assigns a copy, different names for the same list
l2[0] = 4
l1, l2
# ([4,2,3], [4,2,3])
```

This happens for lists, dictionaries, but not for simple data types like int. In this case use a full slice.

3.3 Pitfalls with mutability

We cant make a matrix with the same list repeated because then every row will be a copy. `l.append(x)` modifies `l` in place, returns `None`. `l = l + [x]` creates a new list.

4 August 25, 2022

4.1 Mutability (cotd)

`l1 = l1 + []` is another way to detach `l1` from `l2` where `l2 = l1`.

4.2 Sorting

`l.sort()` sorts by default in ascending in place. `sorted(l)` takes a sorted copy of the list, leaving the original unchanged. Another way to sort inplace will be `list.sort(l)`.

4.3 Equality

`x == y` checks if the values are same, but doesnt check if their memory address is same.

`x is y` checks if both `x` and `y` are pointing to the same location in memory. `x is y` is True implies `x == y` but not vice-versa.

```
l1 = [...]
l2 = l1[:]
l3 = l1
l1 == l2, l1 is l2, l1 == l3, l1 is l3
# True, False, True, True
```

4.4 In functions and parameters

- Pass a mutable value then it can be updated in the function.
- Immutable values will be copied.

List methods are all in place functions. Same for dict methods.

4.5 String

Text, but immutable, like a list, but no single character change possible so immutable. `string[i]` is also a string, and here it is different from a list.

4.6 Slice update in a list

`l1[i:j] = l2`, where `l2` needs to be a list, but not necessarily of the same length. S with this you can grow or shrink the list. This is not necessarily evidently useful for a list, but is useful for strings. However this is actually not allowed for strings.

To do this you have to reconstruct the string. Take the other part of the string apart from the slice to be changed, add the necessary substring, and then update the original string.

4.7 Tuple

Tuples can be used for mutliple assignment. Kind of with tuple upacking. Also, `(a,b) = (b,a)` swaps two values.

4.8 Dictionary

List is a collection indexed by position. It can be thought of as $f : [0, 1, \ldots, n-1] \to [v_0, v_1, \ldots, v_{n-1}]$. A dictionary is a a generalised form of this function, $f : [x_0, x_1, \ldots, x_{n-1}] \to [v_0, v_1, \ldots, v_{n-1}]$. Instead of positions the index is some abstract key. Key can be anything but it **has to be immutable**. So it cannot be a list.

Accessing a nonexistent key results in `KeyError`, analogous to `IndexError` in lists. However assigning to a nonexistent key creates a new key-value pair. This is not possible with index in list.

It is also a mutable collection in the same way a list is. Dictionary cant be indexed like a list, even with multiple keys. You will get `TypeError: 'dict_keys' object is not subscriptable`.

5 August 30, 2022

5.1 Representing sets

A set is not but the collection of keys of a dictionary (in mathematical sense). Maintain a set X from a universe U. then $X \subset U$ is same as a function $X : U \to \{True, False\}$.

Exercise: if `d1` and `d2` both represent sets over some U, compute `d1` $\cup$ `d2`, `d1` $\cap$ `d2`, $U \backslash$`d1`.

5.2 List, array, dict: implementation details

The qs being what are the salient differnces? How are they stored? What is the impact on performance?

5.2.1 Arrays

- contiguous block of memory
- typically size is declared in advance
- same data type for all values, thus every element is of the same size
- `a[0]` typically points to the first value in the allocated block of memory. Hence if I know where `a[0]` is i can quickly go to `a[k]`. Uniform time taken in array
- Array is ***random access***
- *Inserting or deleting*: shift element left/right by 1

5.2.2 List

- not a contiguous block of array
- a sequence of values distributed across memory
- items can be of different data types
- mutable, size not fixed
- ***Sequential access***. Takes time to reach `a[k]` $\propto k$
- *Inserting or deleting*: if we want to delete `a[k]`, we change `a[k-1]`'s pointer to `a[k+1]`. To add, change pointer of `a[k]` to some `x` and then change pointer of `x` to `a[k+1]`. Here by pointer, we dont mean 'pointer' per se, but just that the memory address of the next element in the list.

5.2.3 Dictionary

- key k $\to h(k) \to \{0, 1, \ldots, N-1\}$ where $h(k)$ is some hash function.
- cannot avoid ***collisions*** ($h(k_1) = h(k_2)$). In that case we have to look through multiple elements
- python now lists keys in order of creation
- ***Random access modulo collisions***

6 September 1, 2022

6.1 What are python lists?

Lists are like this: $\boxed{v_k \mid p_{k+1}}$ where v_k is the current element and p_{k+1} is the pointer to the cell containing v_{k+1}. Hence insertion and deletion is easy to implement, but hence computationally list is more expensive.

Python lists are actually arrays. When we initialise an empty list `l`, we are allocated a block of memory of some size for an empty array and a counter pointing to the end of this block which is at 0. **What if `l` grows beyond the size of this array?** It allocates an array of double the size. Thus to exhaust this, we have to do double the amount of work. Why double th amount of work? Because python first has to allocate the array block in memory and then copy the elements from the previous memory block and move it to the new one. While this may seem we are spending a lot of computation power over this set of operation, this is actually only once, and it is balanced out by a large number of computationally cheap operations.

But python lists are not of uniform type like how arrays are. In order to circumvent this problem and yet keep the random lookup feature in python lists, the list (or array) actually stores the lookup addresses of the list elements. So even though the elements themselves may be somewhere else, theres an array with addresses of these elements.

Thus `l.append(x)` is cheap, but `l.insert(idx,x)` is expensive because it involves shifting a lot of elements to the right.

6.1.1 Measuring performance time

We measure performance time using `time.perf_counter()`.

6.1.2 Implementing an actual list using dictionary

The structure should be something similar to this:

```
d = {value : v_0,
     next : {value : v_1
             next : {value : v_2 ...
                          ...{value : v_{n-1}, next : {}}
```

7 September 5, 2022

7.1 List comprehension and other functions

One thing to think about is what is set comprehension.

- Defining new sets from old
- sets contain, a generating set, a filtering conditon, a output transformation. Eg: $\{x^2 \mid x \in \mathbb{Z}, x \geq 0\}$
- more generally, $\{f(x) \mid x \in S, p(x)\}$

We can do this manually for lists as well (using explicit for loops and stuff). But we can use list comprehension too.

We can map functions to every element of the list, or filter it. `map(f, l)` or `filter(p, l)` where `f` is a function applied to every element of `l`, `p` is a *predicate* (returns True if the condition is true else False). Composing these functions allows for the set comprehension logic we showed above.

One thing to note that this does not reduce any time complexity as such. However it is a bit more optimised code, and also is much more readable.

7.2 Inductive definition

1.
   - $0! = 1$
   - $n! = n \times (n-1)!$
2.
   - $\text{fib}(0) = 0$
   - $\text{fib}(1) = 1$
   - $\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2)$

Both use recursion, and define the actual functions by induction.

7.2.1 Recursive function calls

- A function can call itself
- Current executing is suspended until recursion call returns a value, like any other function
- Recursive call will again call itself, so must ensure progress towards a base case for termination

We can also do induction on structures. For example, a list consists of the first element and the rest. Base case for a list is usually `[ ]`. Sometimes there may also be a base case for a singleton element.

8 7 September, 2022

8.1 Evaluation of boolean expressions

- In what order are two expressions compared? By default from left to right.

8.2 Object oriented approach

Defining our own data structures

Can we clearly separate the **interface** from the **implementaton**, and define the data structure in a more modular way? We can. We will use an object oriented approac for that.