Python basics to get started

theroyakash

April 25, 2023

Contents

Chapter 1

Basic Python Introduction

1.1 What is a program?

A program is a sequence of instructions that specifies a computation. It may be something mathematical problem maybe like finding some roots of a polynomial but it also can be a symbolic representation like reading and loading a text into memory from a text file.

For example following is a `text` file, to load it into the memory you might write code like this

```python
file_path = '/users/theroyakash/example.txt'

with open(file_path, mode='r') as f:
    f.read()
```

The details looks different in different languages but a few basic instruction appears in every languages.

For about every language you might do these following operations to complete a specific task in hand meaning completing a computation. Your program may structure like this

- *Input*: Get data from the keyboard, a file, or some other device.
- *Output*: Display data on the screen or send data to a file or other device.
- *Math*: Perform basic mathematical operations like addition and multiplication.
- *Conditional execution*: Check for certain conditions and execute the appropriate sequence of statements.
- *Repetition of small steps*: Perform some action repeatedly, usually with some variation.

Believe it or not, that's pretty much all there is to make a program. Every program you've ever used, no matter how complicated that is, is made up of instructions that look more or less steps like these. We can describe programming as the process of breaking a large, complex task into smaller and smaller subtasks until the subtasks are simple enough to be performed with one of these basic instructions. That may be a little vague, but you will know about this later you study algorithms.

1.1.1 Formal and natural languages

Natural languages are the languages that people speak, such as English, Spanish, and French. They were not designed by people (although people try to impose some order on them); they evolved naturally.

But formal languages are languages that are designed by people for specific applications. For example, the notation that mathematicians use is a formal language that is particularly good at denoting relationships among numbers and symbols. Chemical Scientists use a formal language to represent the chemical structure of molecules. And most importantly: Programming languages are formal languages that have been designed to express computations to be performed on a specialized computing device. Formal languages tend to have strict rules about syntax. For example, $3 + 3 = 6$ is a syntactically correct mathematical statement, but 3=+6$ is not. H2O is a syntactically correct chemical name, but 2Zz is not.

Syntax rules come in two flavors, pertaining to tokens and structure. Tokens are the basic elements of the language, such as words, numbers, and chemical elements. One of the problems with 3=+6*isthat* is not a legal token in mathematics (at least as far as we know). Similarly, 2Zz is not legal because there is no element with the abbreviation Zz. The second type of syntax error pertains to the structure of a statement—that is, the way the tokens are arranged. The statement 3=+6$ is structurally illegal because you can't place a plus sign immediately after an equal sign. Similarly, molecular formulas have to have subscripts after the element name, not before.

1.1.2 First program

Traditionally, the first program written in a new language is called "Hello, World!" because all it does is display the words, "Hello, World!" In Python, it looks like this:

```
print("Hello world, hehe")
```

This is an example of a print statement, which doesn't actually print anything on paper. It displays a value on the screen. In this case, the result is the words `Hello world hehe`.

The quotation marks in the program mark the beginning and end of the value; they don't appear in the result. Some people judge the quality of a programming language by the simplicity of the `\Hello, World!"` program. By this standard, Python does about as well as possible.

Chapter 2

Few cool things

2.1 Order of operations

When more than one operator appears in an expression, the order of evaluation depends on the rules of precedence. Python follows the same precedence rules for its mathematical operators that mathematics does. The acronym PEMDAS is a useful way to remember the order of operations: Parentheses have the highest precedence and can be used to force an expression to evaluate in the order you want. Since expressions in parentheses are evaluated first, $2 * (3 - 1)$ is 4, and $(1 + 1) * *(5 - 2)$ is 8. You can also use parentheses to make an expression easier to read, as in $(\text{minute} * 100)/60$, even though it does not change the result. Exponentiation has the next highest precedence, so $2 * *1 + 1$ is 3 and not 4, and $3 * 1 * *3$ is 3 and not 27. Multiplication and Division have the same precedence, which is higher than Addition and Subtraction, which also have the same precedence. So $2 * 3 - 1$ yields 5 rather than 4, and $2/3 - 1$ is -1, not 1 (remember that in integer division, $2/3 = 0$). Operators with the same precedence are evaluated from left to right. So in the expression $\text{minute} * 100/60$, the multiplication happens first, yielding $5900/60$, which in turn yields 98. If the operations had been evaluated from right to left, the result would have been $59 * 1$, which is 59, which is wrong.

2.2 Comments

Comments As programs get bigger and more complicated, they get more difficult to read. Formal languages are dense, and it is often difficult to look at a piece of code and figure out what it is doing, or why. For this reason, it is a good idea to add notes to your programs to explain in natural language what the program is doing. These notes are called comments, and they are marked with the # symbol, you have seen throughout the book green color comments has been made to inform users on what each line doing.

Chapter 3

If-else statements

Alright let's think of real life: if you have 20 lakh Rupees you would most certainly buy a car or a house in the suburbs. So let's make this statement into code
Let's first store your income into some variables.

```python
myTotalMoney = 2000
# or you can name your variable with (_)s
total_money = 2000
```

Now we need to check whether my total money is greater than 20 lakh so that I can buy a 15 lakh rupees car. So to check this we need to use the python's if keyword (remember in the last chapter).

Let's do that now. To define an if statement we write if then we write conditions for the if block then a colon:
Then in the next line with indentation we write out all the things we want to do in for that condition. So our pseudo-code would look like the following

```python
total_money = 2000

if (...conditions...):
    # Do something
```

So writing this in python would look like the following:

```python
total_money = 2000

if total_money > 2000000:
    # Deduct the money from the total balance
    total_money -= 1500000 # Buying an car of 15 Lakh
    # Now print that the transaction was successful
    print('Transaction Successful')
```

As we don't have money grater than the 20 Lakh mark so our program would not write anything. It'll go to the if statement, seeing the condition doesn't met, it'll do nothing. To tackle this we can write an optional else statement which will be executed when the if block's condition is not met.

```python
total_money = 2000

if total_money > 2000000:
    # Deduct the money from the total balance
    total_money -= 1500000 # Buying an car of 15 Lakh
    # Now print that the transaction was successful
    print('Transaction Successful')
else:
    print('Sorry You do not have 15 lakh rupees so you can not buy
        this car')
```

Your conditions for if statement can be of multiple conditions, you'd join them with and keyword. Let's see an example:
Let's say you want to buy a 15 Lakh car if the car is of TATA brand and you also have money more than 20 lakh in your pocket. So you would write the following code:

```python
total_money = 2000000000
brand = 'TATA'

if total_money > 2000000 and brand == 'TATA':
    # Deduct the money from the total balance
    total_money -= 1500000 # Buying an car of 15 Lakh
    # Now print that the transaction was successful
    print('Transaction Successful')
```

Let's say you have the sufficient money but you want to buy car whether it's TATA or MARUTI and you don't want to buy car if the car is anything other than that so you'll use the or keyword in place of and keyword. So the code will look like the following;

```python
total_money = 2000000000
brand = 'TATA'

if brand == 'MARUTI' or brand == 'TATA':
    # Deduct the money from the total balance
    total_money -= 1500000 # Buying an car of 15 Lakh
    # Now print that the transaction was successful
    print('Transaction Successful')
```

Chapter 4

Introduction to Loops

4.1 What is Loop

Now, let's think about a real life challenge. Suppose you have a number and you have to find whether the number is even or odd? Wait! you know how to work with if-else statements right? So, first think about it. Okay, here how to solve it. You just have to check whether the number is divisible with 2 or not, if the number is divisible by 2 then the number is even, otherwise the number will be odd, simple right? Now, suppose you have given a sequence of numbers and now you have to find the even numbers from that sequence. Here comes the factor called loop. A loop is used to iterate over a sequence, where the sequence can be list, tuples, dictionary, set or string. You'll find what those are in the following chapters.

So, now let's implement a loop in order to solve this problem. First we have to traverse over the sequence of numbers and have to check each element whether that particular element is an even, if so then we have to insert that element into the final list and after finishing the iteration, the final list will be the output of our question.

4.2 Types of Loops

There are two types of loops first is 'for' loop and other is 'while' loop. Using which we'll implement the upper problem.

Here's the python code with for loop

```python
given_list_of_element = [6, 9, 2, 1, 8]
final_list = []

for i in given_list_of_element:
    if i%2==0:
        final_list.append(i)

print(final_list) # [6, 2, 8]
```

Here's the python code with while loop

```python
given_list_of_element = [6, 9, 2, 1, 8]
final_list = []
i = 0

while i < len(given_list_of_element):
    if given_list_of_element[i]%2==0:
        final_list.append(given_list_of_element[i])

    i+=1 # Increase i so that the while loop doesn't get stuck in a
        forever loop

print(final_list) # [6, 2, 8]
```

Chapter 5

Arrays and Lists

5.1 What is array?

Until this moment what we did here was that we stored some values in some location and gave that location a name to reference that stored data. We called this variables. What if we wanted to store more than one thing like sequence of numbers or list of all students in a variable?

Now we implement arrays (Python doesn't have builtin support for this instead we use lists here). The difference between arrays and list is that list can hold values of multiple types meaning it can store a integer and a string and a float together. But for arrays, they can only store a single type of data. In python you'll define a list like this

```python
list_of_elements = [1, 2, 3, 4, 5, "Hello World"]
```

5.2 Accessing elements from a list

We can access specific elements from a list. We have to specify an index(position) from where we want to see the values. These indexes starts from zero in python. You need to use the index operator `[ ]` to access an item in a array. The index must be an integer. Now to access the 0th element from the variable `list_of_elements` We'll write the following code and it'll return `1`

```python
print(list_of_elements[0])   # --> This will print 1
```

To see how many elements you have in your array you can use a builtin function `len()`. There is a shortcut to quickly accessing the last element in the list: to use `[-1]` index.

```python
print(list_of_elements[-1])   # --> This will print "Hello World"

# Returning the size of the list
print(len(list_of_elements))  # --> This will print 6
```

5.3 Inserting into lists

Inserting into lists are super easy to do. There is a built in function called `.append()`, when you call this function on a list it'll add new numbers to the end of the list. Here is how you will do it

```python
list_of_elements = [1, 2, 3, 4, 5, "Hello World"]
list_of_elements.append(7)

print(list_of_elements)  # --> This will print [1, 2, 3, 4, 5, "
    Hello World", 7]
# The length of the list_of_elements will also increase by 1.
print(len(list_of_elements))  # --> 7
```

5.4 Deleting a specific item from a list

Just like inserting deleting also a very easy task to do. Just use the keyword `del` in front of the index to delete the element from the index. Here's how you need to do it

```python
list_of_elements = [1, 2, 3, 4, 5, "Hello World"]
del list_of_elements[5]

print(list_of_elements)  # --> This will print [1, 2, 3, 4, 5]
# The length of the list_of_elements will also decrease by 1.
print(len(list_of_elements))  # --> 5
```

5.5 Updating a specific element in list

To update a specific element you can just reassign the element to the new one. Let's say we need to change the 1st element (*0th element is the 1st, and 1st element is the 2nd as python has zero based indexing*), you will use the following code:

```python
list_of_elements = [1, 2, 3, 4, 5]
list_of_elements[0] = 102

print(list_of_elements)  # --> This will print [102, 2, 3, 4, 5]
```

Chapter 6

Announcement

6.1 Currently and in the future

Hey guys I have a quick announcement to make. This version of the basic python book is the newest edition. In the next version I'll include for loops, concepts of simple data structures like arrays linked list etc.

I'm trying very hard to come up the full version for this book. New chapters like Functions, Object oriented programming is in the pipelines. I'm currently a CS undergraduate and developing useful libraries for python. So it's really hard for me to manage the book and update you regularly.

If you want to connect with me or if you have any suggestions regarding this book I'd love to connect with you. Send me a message via Twitter @theroyakash or connect with me on instagram @theroyakash or LinkedIn at theroyakash.

My github profile is @theroyakash and I'm available on discord for direct messaging at @theroyakash#5635.