Lecture Notes on Language Basics

15-112: Introduction to Programming and Computer Science
Saquib Razak

Lecture 2
August 28, 2019

1 Language Basics

Most computer programs take input from users, process this input and then produce some kind of output. In order to write computer programs, we will need to learn how to read input from users, save input in variables, process this input using expression and then output the result to the screen. In this lecture, we will learn the following topics:

- Printing to the screen
- Expressions and types
- Storing values in Variables
- Reading values from the users

2 Output: Printing to the screen

In this section, we will learn how to use the print statement to produce output on the screen. Following is an example of printing a string:

```
print ("Python Rocks!!")
```

The above statement will put the string "Python Rocks!!" on the screen. Each print statements prints one complete line and moves the cursor to the next line. You can print multiple values on the same line by separating them by commas ',', All values separated by commas, are printed on the same line with a single space between each value. (Note: Any text in your code followed by a # sign is ignored by Python until the end of that line. These are called comments and are used to document your code.)

```
print (9)
print (4.5)
print ("Ahmed","Abdullah")    # This will print Ahmed Abdullah to the screen
print (9,4.5,"Ali")           # This will print 9 4.5 Ali
```

The output of the above program would be

```
9
4.5
Ahmed Abdullah
9 4.5 Ali
```

3 Types

Each value in python has a type. In the expression $2 + 3$, there are two values 2 and 3 and each of which is of type integer. Integers are positive or negative numbers that do not have any fractional parts (decimal points). You can print integers on the screen by using the print statement:

```
print 45
print 33
print (17,18,19)
```

The above statements will print 45 and 33 on separate line followed by 17 18 19 on a single line.

Numbers that have decimal points are of type ”floating point numbers” or simply ”float”. For example 3.4 is a floating point number since it has a decimal point. 19.0 is also of type float.In python, floats can also be represented in the scientific notation. For example: 3.9E15 represents the value $3.9 * 10^{15}$ and 1.0E-6 represents $1 * 10^{-6}$ which is equal to 0.000001. Again, print statement can be used to print floats on the screen.

Another common type for values is a $string$. Strings are a combination of ”typeable” (characters you can type using a keyboard) enclosed within single quotes (’) or double quotes (”). Here are some example of strings:

- ”Hello World”
- ’Hello World !!’
- ”Cap’n Crunch”
- ’ I said : ”Oh be quiet!!” ’
- ’ What is 3 + 4 ? ’

There is another type in python that represents a true or false value. This type is called Boolean. We will study this more when we look at expressions in detail. For now, I will just state the obvious, Boolean types can only have two values $True$ or $False$.

4 Expressions

An expression is a combination of operators and operands that results in a single value. $2 + 3$ is a simple expression that consists of two operands: 2 and 3 and one operator $+$ and results in a single value 5.

As a principle, when you add two integer values, you get a result that is of type integer as well (with the exception of the divide operator). If you combine integers with floating point numbers, the result will always be a float. For example:

$9 + 3.5$ will result in 12.5 which is of type float.

Similarly, $2.5 + 4.5$ will result in 7.0 also of type float.

Before we look at more examples of expressions, lets first study different types of operators available in python.

4.1 Operators

We are used to many different operators for processing numbers. Python provides most of these operators. Following is a list of operators we are already familiar with. These operators are called "Arithmetic Operators":

- $+$ (Addition operator)
- $-$ (Subtraction)
- $*$ (Multiplication)
- / (Division)
- // (Integer Division)

The first three operators $+, -, *, /$ work as expected. The integer division operator is a little bit tricky. In python, dividing two integers using the integer divide (//) operator gives you the quotient of the division and any remainder is ignored. Quotient is the number of times the divisor divides into the dividend evenly. Maybe an example will help make more sense of the previous statement:

Let's try to divide 29 by 4. Here if we do the division we have learned in Math, the quotient is 7 and the remainder is 1. The integer division in python will result in 7 and the remainder would be ignored

The following table shows some more examples that will help illustrate the concept:

Expression	Value
6 // 3	2
8 // 2	4
15 // 7	2
14 // 7	2
16 // 7	2
3 // 6	0
1 // 2	0

You can try these expressions by creating a new python file or using an existing one and typing the following commands:

```
print 6//3
print 8//4
print 15//7
print 14//7
print 16//7
print 3//6
print 1//2
```

If you save and run the above program, you should see the same results as given in the table above.

Similar to the divide operator, python (and most programming languages) have an operator to find the remainder of a division operation. This operation is called the ”modulus” operation - also referred to in the short form ”mod” and is represented by the ”%” symbol. For Example:

Expression	Value
6 % 3	0
15 % 7	1
14 % 7	0
16 % 7	2
3 % 6	3
1 % 2	1

4.1.1 Comparison Operators

I mentioned earlier that python has a Boolean type that results in either a true or a false value. The comparison operators in python produce boolean results.

- $>$ greater than

- $<$ less than
- $>=$ greater than or equal to
- $<=$ less than or equal to
- $==$ equal to
- $!=$ not equal to

Here are some examples of using these comparison operators. The usefulness of these operators will be apparent when we study condition execution later in the semester.

- print $3 > 4$ # will print False
- print $3 < 4$ # will print True
- print $3 <= 4$ # will print True
- print $4 >= 4$ # will print True
- print $4 != 4$ # will print False
- print $7 == 7$ # will print True

4.1.2 Bitwise Operators

Python also has bitwise operators like "|" representing OR, "&" representing AND, and ^ representing Exclusive OR operators. For these and more bitwise operators, please read pages 18 and 19 from your Text Book - "Exploring Python".

4.2 Operator Precedence

We mentioned earlier that expressions are a combination of operators and operations that result in a single value. An expression can have more than one operators - for example:

$1 + 2 * 3$

What is the sequence of evaluation of these operations? As you can see, the two different ways of looking at this expression will result in very different values.

```
(1 + 2) * 3
```

or

```
1 + (2 * 3)
```

The rules that determine the order in which operations are performed are called operation precedence. In the above example without parenthesis, the multiplication will be done before addition since multiplication has a higher precedence than addition. The result of the expression $1 + 2 * 3$ would be 7. In general, multiplication (division is a kind of multiplication) has a higher precedence than addition (subtraction is a kind of addition). In an expression that has operations with same precedence, the operations will be performed from left to right. The operations within parenthesis will always be performed first.

Here are a few examples, Can you reason about the values of these expressions?

Expression	Value
3 + 4 - 6	1
3 + 4 * 6	27
(3 + 4) * 6	42
2 * 4 + 1	9
2 * 4 // 3	2
8 // 2 * 3	12
8 // (2 * 3)	1
2 // 4 * 3	0
2 / 4 * 3	1.5

Some more complicated examples are presented below. Study the value of the expression and make sure you can explain the sequence of operations:

Expression	Value
3 + 4 * 2 - 5	6
(3 + 4) * (2 - 5)	-21
1 // 2 * 8	0
8 * 1 // 2	4
8 * (1 // 2)	0
8 * (1 / 2)	4.0
3 + 4 * 5 // 2	13
3 * 4 - 2 * 5	2
3 * 4 * 2 - 1	23

5 Variables

Computer programs deal with a lot of data. Computers store data as values in memory. We can access these values by using symbolic names called "variables". Values can be assigned to variables by using

the assignment operator "=". The assignment operator "=" is read as "is assigned" in an expression. For example, we can store the value 18 in a variable called age as follows:

```
age = 18
```

This statement is read as "age is assigned 18". From here on, the value of variable "age" would be 18 until another assignment statement modifies this value. We can print the value of a variable by using the print statement.

```
age = 18
print (age)
print ("your age is ", age )
print ("half of your age is " , age//2)

The output of the above program would be:
18
your age is 18
half of your age is 9
```

We can change the value of a variable by assigning it a new value:

```
age = 19
print ("My age is ", age)          #variable age has value 19
print ("Let's assume that a year has already passed")
age = 20
print ("Now my age is ",age)       #now variable age has value 20
```

We can use variables in expressions. Whenever we have a variable on the left side of the "=" operator, we end up writing a new value to the variable. Any other use of the variable is almost always only reading the value of the variable but does not change the value. Following examples illustrate some of these ideas:

```
age = 18       # represents how old a person is in years
ageinMonths = age * 12    # number of months in age years
ageinDays = ageinMonths * 30    # assuming 30 days per month(an inaccurate assumption)
ageinHours = ageinDays * 24
ageinMinutes = ageInHours * 60

age = 21       # since the programs execute sequentially,
               # age changes to 21 but the rest of the varaibles don't
```

6 Input

So the last thing that we will cover in this lecture is reading input from the user. This will complete all the components that allow us to read input from the user, process this input, and produce some sort of output.

Whenever we read a value from the user, we need to save it in a variable so we can use it later. The easiest way to read a string from the user is to use a function called *input*. In the code shown below, the program execution will pause at the statement shown until the user types something and then presses the "Enter" key. All text entered by the user will be saved in the variable called "name". Printing this variable will print the input entered by the user:

```
name = input()
print ("You entered", name)
```

It is common that user input is preceded by some text that prompts the user to answer a question. It would be more instructive to have our program print "Please enter your name " if we want the user to enter his/her name. We can do that by passing the message string as an input parameter to the *input* function as shown below:

```
firstName = input("Please enter your first name ")
```

The function "*input*" only reads strings. What if we want to read an integer? In order to read integers, floats, or booleans from the user, you need to read the input as a String and then convert it to the appropriate type. There are built-in functions that you can use for this conversion. For example, int() will convert a string passed in to an integer as shown in the program below:

```
firstName = input("Enter your first name: ")
 # Here we will read a string and then pass it to int function to convert it to integer
age = int(input("Enter your age: " ))
```

Similarly the function bool() will convert an input string to boolean, and float() will convert a string to float. The following examples illustrate that point - Remember if the user enters a value that is not a valid integer, float or boolean and you try to convert it, the program will crash:

```
# Now we will read gpa as a string and convert it to float
gpa = float(input("Enter your gpa: "))

# Lastly, we will read a boolean representing the student standing
freshmen = bool (input("Are you a freshmen?  Answer True or False"))
```