IIT Madras
BSc Degree

Home Lesson-2.3

Lesson-2.2

Input

Accepting input from the user routinely happens in programming. Any piece of software shipped to a customer needs to have a functional interface that will let the user interact with the software. We all have used apps like Facebook, Instagram and Twitter. These apps regularly accept input from the user, though we seldom look at it from a programming perspective. Take the case of commenting on a post in Facebook. The text entered in the comment-box is the input. The code running in the backend processes this input and then displays it as a comment in a visually appealing form.

Python provides a built-in function called `input()` to accept input from the user. This is simple yet powerful:

```
x = input()
print('The input entered by the user is', x)
```

Execute the code given above and head to the console. Here the interpreter waits patiently for you to enter text. Press `enter` after entering the input. This acts as a cue for the interpreter to understand that you have completed entering your input. This text is stored in the variable `x`. The way it looks in the console is as follows:

```
1
The input entered by the user is 1
```

Sometimes we may want to prompt the user to enter a particular type of input. This can be done by passing the instruction as an argument to the input function:

```
x = input('Enter an integer between 0 and 10: ')
print('The number entered by the user is', x)
```

Let us now look at the type of the variable `x`:

```
x = input()
print('The input entered by the user is of type', type(x))
```

Execute the above code with the following input types: `int`, `float`, `str` and `bool`. What is the output in each case? We see that the `input()` function always returns a string. Even if the user enters a number, say `123`, that is processed as the string `'123'`. If we want to accept an integer as input, how do we do it? We take the help of an operation called type conversion.

Type Conversion

If we want to convert a string into an integer, Python provides a built-in function called `int`:

```
x = '123'
print('The type of x is', type(x))
y = int(x)
print('The type of y is', type(y))
```

The operation in line-3 is called type conversion, i.e., we are converting an object of type `str` into an object of type `int`. The inverse operation also works. Predictably, the function needed for this purpose is called `str`:

```
x = 123
print('The type of x is', type(x))
y = str(x)
print('The type of y is', type(y))
```

If we want to accept an integer input from the user, we first take a string as input and then convert it into an integer:

```
x = input('Enter an integer: ')
x = int(x)
print('The integer entered by the user is', x)
```

Instead of writing this in two lines, we could write this in a single line:

```
x = int(input())
print('The integer entered by the user is', x)
```

What we have done in line-1 is to compose two functions. That is, pass the output of the inner function - `input()` - as the input of the outer function - `int()`. In the above code, what happens if the input entered is a float value?

```
x = int(input())    # user enters a float value here
```

The code will throw a `ValueError`. Let us take a concrete example. When the command `int('1.23')` is entered, the interpreter tries to convert the string `'1.23'` into an integer. But the number enclosed within the quotes is not an `int`, but a `float`. This number cannot be converted into an integer, hence the error.

Built-in Functions

We have been using the term `built-in functions` quite often. These are functions that have already been defined. Loosely speaking, a function in `Python` is an object that accepts inputs and produces outputs. For example, `print` is a built-in function that accepts an input and prints it to the console.

We will look at few more functions which will come in handy.

- `round` accepts a number as input and returns the integer closest to it. For example, `round(1.2)` returns `1`, while `round(1.9)` returns `2`.
- `abs` accepts a number as input and returns its absolute value. For example, `abs(-1.2)` returns `1.2`.
- `int` is a bit involved. If an integer enclosed within quotes (string) is entered as input, then the output is that integer. We have already seen this: `int('123')` is `123`. If a float is entered as input, then the decimal part is thrown away and the integer part is returned. For example, `int(1.2)` returns `1` and `int(-2.5)` returns `-2`.
- `pow` is another useful function. `pow(x, y)` returns the value of x^y. This performs the same function as the `**` operator. In general, the `**` operator is faster than the `pow` function. But for small numbers, the difference is not perceptible. In fact, using the `pow` function increases readability of code. An extra feature of `pow` is that it supports a third argument: `pow(x, y, z)` returns the value of $x^y \mod z$. That is, it gives the remainder when x^y is divided by z.
- `isinstance` is used to check if an object is of a specified type. For example `isinstance(3, int)` returns the value `True` as the literal `3` is of type `int`. The first argument could be any object, not just a literal. For example, if `x` is a variable of type `str` then, `isinstance(x, str)` will again return `True`.

The Python documentation provides an exhaustive list of built-in functions.