

Solution Review: String to Integer

This lesson contains the solution review to convert a string into an integer in Python.

We'll cover the following



- Implementation
- Explanation

In this lesson, we will review the solution to the challenge in the last lesson. The problem is as follows:

Given some numeric string as input, convert the string to an integer.

You already know that we are not allowed to use the built-in `int()` function. Now to convert a string into an integer, we can make use of the following arithmetic:

$$\begin{array}{c} 123 \\ \downarrow \\ 1*10^2 + 2*10^1 + 3*10^0 \\ \downarrow \\ 100 + 20 + 3 = 123 \end{array}$$

As you can see, by multiplying each digit with the appropriate base power, we can get the original integer back. Let's implement this in Python.

Implementation

```
1 def str_to_int(input_str):
2
3     output_int = 0
4
5     if input_str[0] == '-':
6         start_idx = 1
7         is_negative = True
8     else:
9         start_idx = 0
10        is_negative = False
11
12    for i in range(start_idx, len(input_str)):
13        place = 10**(len(input_str) - (i+1))
14        digit = ord(input_str[i]) - ord('0')
15        output_int += place * digit
16
17    if is_negative:
18        return -1 * output_int
19    else:
20        return output_int
21
22
23
24 s = "554"
25 x = str_to_int(s)
26 print(type(x))
27
28 s = "123"
29 print(str_to_int(s))
30
31 s = "-123"
```



Explanation

On **line 3**, `output_int` is initialized to `0`. The code on **lines 5-10** deal with the polarity of the number represented by `input_str`:

```
if input_str[0] == '-':
    start_idx = 1
    is_negative = True
else:
    start_idx = 0
```

```
start_idx = 0  
is_negative = False
```

If the first character of `input_str` is `-`, `start_idx` is set to `1` and `is_negative` is set to `True` (lines 6-7). `start_idx` is to indicate that we'll start processing `input_str` from the first index as `-` is on the zeroth index. However, if `input_str[0]` is not equal to `-`, `start_idx` and `is_negative` are set to `0` and `False` respectively on lines 9-10.

Next, let's discuss the `for` loop on line 12:

```
for i in range(start_idx, len(input_str)):  
    place = 10**(len(input_str) - (i+1))  
    digit = ord(input_str[i]) - ord('0')  
    output_int += place * digit
```

This `for` loop will run from `start_idx` to `len(input_str) - 1`. On line 13, `place` is set to `10` raised to the power of `len(input_str) - (i+1)`. For example, if `input_str` is `123`, `place` will have the following values:

```
input_str = 123  
len(input_str) = 3  
  
i = 0  
place = 10** (3 - (0+1)) = 10** (2) = 100  
i = 1  
place = 10** (3 - (1+1)) = 10** (1) = 10  
i = 2  
place = 10** (3 - (2+1)) = 10** (0) = 1
```

You may notice that `place` is the base power calculated according to the place of the digit in the number. For the unit place, `place` will equal `1`. Once we have calculated the base power, we need to extract the digit as a number. Therefore, on line 13, we get the digit as an integer type by subtracting the Unicode code point of `0` from the Unicode code point of `input_str[i]`. The Unicode code points are calculated using the `ord` function.

Finally, all that is left for us is to multiply `digit` with `place` and add it to `output_int` which is done on line 15.

Lines 17-20 return `output_int` with its corresponding polarity.

```
if is_negative:  
    return -1 * output_int  
  
else:  
    return output_int
```

If `is_negative` is `True`, `output_int` is returned after multiplication with `-1` on **line 18**; otherwise, it is returned as it is on **line 20**.

That's all, folks! Now we have come to an end to the course. I hope you had an amazing learning experience!