Integer to String

In this lesson, you will learn how to convert an integer to a string in Python.

In this lesson, we will solve the following problem:

You are given some integer as input, (i.e. … -3, -2, -1, 0, 1, 2, 3 …) and you have to convert the integer you are given to a string. Examples:

```
Input: 123
Output: "123"
Input: -123
Output: "-123"
```

Note that you cannot make use of the built-in `str` function:

```python
print(str(123))
print(type(str(123)))
```

Before diving into the implementation of the solution, we need to get familiar with the following functions:

1. `ord()`
2. `chr()`

You might be able to recall `ord()` from one of the previous lessons. `ord()` returns an integer which represents the Unicode code point of the Unicode character passed into the function. On the other hand, `chr()` is the exact

opposite of `ord()`. `chr()` returns a character from an integer that represents the Unicode code point of that character.

```
## Prints 48 which is the Unicode code point of the character '0'
print(ord('0'))
## Prints the character '0' as 48 is Unicode code point of the character '0'
print(chr(ord('0')))
## Prints 49
print(ord('0')+ 1)
## Prints 49 which is Unicode code point of the character '1'
print(ord('1'))
## Prints the character '2' as 50 is Unicode code point of the character '2'
## ord('0') + 2  = 48 + 2 = 50
print(chr(ord('0')+ 2))
## Prints the character '3' as 51 is Unicode code point of the character '3'
## ord('0') + 3  = 48 + 2 = 51
print(chr(ord('0')+ 3))
```

From the above coding example, you can observe the following pattern:

```
ord('0') = 48
ord('1') = ord('0') + 1 = 48 + 1 = 49
ord('2') = ord('0') + 2 = 48 + 2 = 50

chr(ord('0')) = chr(48) = '0'
chr(ord('0') + 1) = chr(48 + 1) = chr(49) = '1'
chr(ord('0') + 2) = chr(48 + 2) = chr(50) = '2'
```

At this point, you must be familiar with the workings of `ord` and `chr` functions. Let’s discuss the actual implementation now.

Implementation

```
def int_to_str(input_int):

    if input_int < 0:
        is_negative = True
        input_int *= -1
    else:
        is_negative = False

    output_str = []
    while input_int > 0:
```

```
    while input_int > 0:
        output_str.append(chr(ord('0') + input_int % 10))
        input_int //= 10
    output_str = output_str[::-1]

    output_str = ''.join(output_str)

    if is_negative:
        return '-' + output_str
    else:
        return output_str


input_int = 123
print(input_int)
print(type(input_int))

output_str = int_to_str(input_int)
print(output_str)
print(type(output_str))
```

int_to_str(input_int)

Explanation

From **line 3** to **line 7**, we determine whether `input_int` is a negative or a positive integer:

```
if input_int < 0:
        is_negative = True
        input_int *= -1
    else:
        is_negative = False
```

If `input_int` is less than `0`, `is_negative` is set to `0` on **line 4** and is also converted to a positive integer by multiplication with `-1` on **line 5**. On the other hand, if `input_int` is a positive integer, `is_negative` is set to `False` on **line 7**.

Once the sign of `input_int` is determined, the execution jumps to **line 9**. Here is the code from **lines 9-13**:

```
    output_str = []
    while input_int > 0:
        output_str.append(chr(ord('0') + input_int % 10))
        input_int //= 10
    output_str = output_str[::-1]
    output_str = output_str[::-1]
```

`output_str` is initialized to an empty list on **line 9**. The `while` loop on **line 10** is where the actual action happens. The last digit of `input_int` is extracted using the `%` operator on **line 11**.

For example,

```
12 % 10 = 2
17 % 10 = 7
10 % 10 = 0
```

You notice that by taking the modulus of a number with `10`, we always get the last digit of that number. The last digit is converted into a character by `chr()` and `ord()` functions. So `ord('0') + input_int % 10` gives the Unicode code point of the character that we want which when passed to `chr()` function returns a character on **line 11**. That character is appended to `output_str` in the same line. On **line 12**, as we have already dealt with the last digit, we remove it from `input_int` using the `//` operator. The `while` loop repeats the code on **lines 11-12** until `input_int` becomes less than or equal to `0`.

On **line 13**, `output_str` reverses the positions of its elements so that the last digit is on the last index of `output_str` instead of being on the first one. That brings us to the code on **lines 15- 20**:

```
output_str = ''.join(output_str)

    if is_negative:
        return '-' + output_str
    else:
        return output_str
```

On **line 15**, the elements in `output_str` are joined together using the `join` function. The `''` acts as a separator which translates to the fact that there will be nothing between the elements as `''` is just an empty string. Now that we have `input_int` converted to an integer, the only thing left for us is to deal with its sign. If `is_negative` is `True`, we add a `-` before `output_str` and return it on **line 18**, otherwise `output_str` is returned without any further modification on **line 20**.

You can confirm the type of outputs from the built in `type` function in Python

You can confirm the type of outputs from the built-in `type` function in Python.

So that was how we convert an integer to a string in Python. Interesting, right? Now get ready for a challenge in the next lesson where you’ll have to convert a string into an integer — excited, right? See you there!