Python Integers

Summary: in this tutorial, you'll learn about Python integers and how Python stores integers in the memory.

Integers are whole numbers that include negative numbers, zero, and positive numbers such as -3, -2, -1, 0, 1, 2, 3.

Python uses the class `int` to represent all integer numbers. All integers are objects.

How computers store integers

Computers can't store integers directly. Instead, they only can store binary numbers such as 0 and 1.

To store integers, the computers need to use binary numbers to represent the integers.

For example, to store the number 5, the computers need to represent it using a base-2 number:

$5 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$

As you can see, it takes 3 bits to store the number 5 in the memory:

$(101)_2 = (5)_{10}$

Suppose that you have 8 bits, you can store up to 255 integers from zero to 255:

$255 = 1x\ 2^7 + 1\ x\ 2^6 + 1\ x\ 2^5 + 1x\ 2^4 + 1\ x\ 2^3 + 1\ x\ 2^2 + 1x\ 2^1 + 1\ x\ 2^0$

By using 8 bits, you can store up to $2^8 - 1 = 255$ integers.

To store both negative integers, zero, and positive integers, you need to reserve 1 bit for storing the sign, negative (-) and positive (+). Therefore, with 8 bits:

- The largest integer that the computers can represent is $2^7 = 127$.
- And the computers can store all integers in the range (-127, 127)

Because the number zero doesn't have a sign, the computers can squeeze out an extra number. Therefore, 8 bits can store all integers from -128 to 127.

8-bits = $[-2^7, 2^7 - 1]$

Similarly, if you want to use 16 bits, 32 bits, and 64 bits to store integers, the ranges would be:

- 16-bits ~ $[-2^{15}, 2^{15} - 1]$ = [`-32,768` , `32,767`]
- 32-bits ~ $[-2^{31}, 2^{31} - 1]$ = [- `2,147,483,648` , `2,147,483,647`]
- 64-bits ~ $[-2^{63}, 2^{63} - 1]$ = [`-9,223,372,036,854,775,808` , `9,223,372,036,854,775,807`]

How Python represents integers

Other programming languages such as Java and C# use a fixed number of bits to store integers.

For example, C# has the `int` type that uses 32-bits and the `long` type that uses 64 bits to represent integers. Based on the integer types, you can determine the ranges of the integers those types can represent.

Python, however, doesn't use a fixed number of bit to store integers. Instead, **Python uses a variable number of bits to store integers**. For example, 8 bits, 16 bits, 32 bits, 64 bits, 128 bits, and so on.

The maximum integer number that Python can represent depends on the memory available.

Also, integers are objects. Python needs an extra fixed number of bytes as an overhead for each integer.

It's important to note that the bigger the integer numbers are, the slower the calculations such as `+`, `-`, ... will be.

Python int type

The following defines a variable (https://www.pythontutorial.net/python-basics/python-variables/) that references (https://www.pythontutorial.net/advanced-python/python-references/) an integer and uses the `type()` function to get the class name of the integer:

```
counter = 10
print(type(counter))
```

Output:

```
<class 'int'>
```

As you can see clearly from the ouput, an integer is an instance of the class `int`.

Getting the size of an integer

To get the size of an integer, you use the `getsizeof()` function of the `sys` module.

The `getsizeof()` function returns the number of bytes that Python uses to represent an integer. For example:

```
from sys import getsizeof

counter = 0
size = getsizeof(counter)

print(size)  # 24 bytes
```

Ouput:

```
24
```

To store the number 0, Python uses 24 bytes. Since storing the number zero, Python needs to use only 1 bit. Note that 1 byte equals 8 bits.

Therefore, you can think that Python uses 24 bytes as an overhead for storing an integer object.

The following returns the size of the integer 100:

```
from sys import getsizeof

counter = 100
size = getsizeof(counter)

print(size)  # 28 bytes
```

Output:

```
28
```

It returns 28 bytes. Since 24 bytes is an overhead, Python uses 4 bytes to represent the number 100.

The following shows the size of the integer 2^{64} :

```
from sys import getsizeof

counter = 2**64
size = getsizeof(counter)

print(size)  # 36 bytes
```

Output:

```
36
```

So to store the integer 2^{64}, Python uses 36 bytes.

Python integer operations

Python integers support all standard operations including:

- Addition +
- Subtraction –
- Multiplication *
- Division /
- Exponents **

The result of addition, subtraction, multiplication, and exponents of integers is an integer. For example:

```
a = 10
b = 20

c = a + b
print(c)
print(type(c))


c = a - b
print(c)
print(type(c))


c = a * b
print(c)
print(type(c))
```

```
c = a ** b
print(c)
print(type(c))
```

Output:

```
30
<class 'int'>
-10
<class 'int'>
200
<class 'int'>
100000000000000000000
<class 'int'>
```

However, the division of two integers always returns a floating-point number. For example:

```
a = 10
b = 5
c = a / b

print(c)
print(type(c))
```

Output:

```
2.0
<class 'float'>
```

Summary

- Integers are whole numbers that include negative whole numbers, zero, and positive whole numbers.

- Computers use binary numbers to represent integers.
- Python uses a variable number of bits to represent integers. Therefore, the largest integer number that Python can represent depends on the available memory of the computer.
- In Python, all integers are instances of the class `int` .
- Use the `getsizeof()` function of the `sys` module to get the number of bytes of an integer.
- Python integers support all standard operations including addition, subtraction, multiplication, division, and exponent.