Лабораторна робота №3
Представлення чисел у двійковому вигляді та побітові оператори.
Короткі теоретичні відомості.

Комп'ютери працюють з двійковими числами. Одне двійкове число це біт.

Нібл - чотирибітне утворення. Нібл містить 4 біти, і це відповідає шістнадцять (2^4) можливих значень, отже нібл відповідає одному шістнадцятковому числу (тому нібл часто називають «шістнадцяткове число».

Для сучасного комп'ютера найменша одиниця даних, з якими він працює - це байт, який складається з вісьми бітів (двійкових чисел). Це означає, що цілі числа від 0 ... 255 можуть бути представлені в одному байті.

Два байти часто називають словом. Так само словом можуть називати іноді чотири байти або вісім, або і більше значення.

Наприклад, число 211 це в двійковій системі числення це вісім біт 11010011, два нібли D (1101 в шістандцятковій системі) та 3(0011 в шістандцятковій системі) і для його представлення достатньо одного байта.

Представленння цілих чисел

В Python крім десяткової можна використовувати двійкову, вісімкову та шістнадцяткову системи числення.

Для створення числа в двійковій системі перед його значенням потрібно вказати 0 та префікс b:

```
x_bin = 0b111 # 111 у двійковій системі це 7 в десятковій
```

Для створення числа у вісімковій системі перед його значенням потрібно вказати 0 та префікс o:

```
y_oct = 0o11  # 11 у вісімковій системі це 9 в десятковій
```

Для створення числа в двійковій системі перед його значенням потрібно вказати 0 та префікс x:

```
z_hex = 0x0b    # b у шістнадцятковій системі це 11 в десятковій
```

Незалежно від того, яка система використовувалася для запису числа, з числами можна проводити всі відомі арифметичні операції:

```
>>> x_bin = 0b111 # 7
>>> y_hex = 0x0b   # 11
>>> x + y         # 18
18
>>> x/y
0.6363636363636364
>>> x**y
1977326743
```

Вбудовані функції bin(), oct(), hex() дозволяють перетворювати числа з однієї системи числення в іншу. Потрібно тільки звернути увагу, що результат перетворення це рядок символів, які відповідають числу в іншій системі числення. При спробі виконати арифметичні операції, насправді будуть виконуватися дії, які властиві типу str.

```
>>> bin(11)
'0b1011'
>>> oct(11)
'0o13'
>>> hex(11)
'0xb'
>>> hex(11) + bin(11)
'0xb0b1011'
```

Можливі також і зворотні перетворення рядка символів, які відповідять числу в одній із систем числення до типу int. Для цього використовується вбудована функція int() але з двома аргументами. Перший аргумент це рядок символів a, другий це система числення з якої потрібно зробити перетворення.

```
>>> int(hex(11), 16)
11
>>> int(oct(11), 8)
11
>>> int(bin(11), 2)
11
```

Зазвичай при використанні типу int в програмах цілі числа записують в десятковій системі. В пам’яті комп’ютера ці числа будуть представлені у двійковому вигляді. В залежності від того скільки байтів виділяється для збереження цілого числа залежить яке максимальне число можна зберегти. В сучасних комп’ютерах для збереження цілих чисел можна використовуати 8 байтів (64 біти) і таким чином максимальне число, яке можна зберегти буде 18446744073709551615 (2**64-1). Для операцій з більшими числами потрібно використовувати спеціальні підходи. В Python реалізовано механізм, який дозволяє працювати з як завгодно великими цілими числами й від користувача приховано перехід від звичайного пердставлення цілого числа до представлення великого простого числа. За допомогою модуля sys можна дізнатися яке максимальне ціле число можна представити на вашому комп’ютері в Python.

```
>>> import sys
>>> sys.maxsize
9223372036854775807
```

Звертання до sys.maxsize повертає максимальне значення цілого числа. Зазвичай 2 ** 31 - 1 на 32-розрядній платформі та 2 ** 63 - 1 на 64-розрядній платформі.

Крім арифметичних операторів до цілих чисел в Python, як і в інших мовах програмування можна застосовувати побітові оператори. В таблиці наведено перелік побітових операторів та приклади виконання побітових операцій.

Синтаксис	Розширений оператор	Опис операції над числами	Приклад
i & j	&= i = i & j	Побітна операція І (AND) 0 &0 → 0 0 &1 → 0 1 &0 → 0 1 &1 → 1	5 & 3 дає 1 0b101&0b011 → 0b001 101 011 001
i | j	|= i = i | j	Побітна операція АБО (OR) 0 | 0 → 0 0 | 1 → 1 1 | 0 → 1 1 | 1 → 1	5 | 3 дає 7 0b101|0b011 → 0b111 101 011 111
i ^ j	^= i = i ^ j	Побітна операція ВИКЛЮЧНО АБО (XOR) 0 ^ 0 → 0 0 ^ 1 → 1 1 ^ 0 → 1 1 ^ 1 → 0	5 ^ 3 дає 6 0b101^0b011 → 0b110 101 011 110
~i		Побітне НЕ (NOT) x це -(x+1) ~0 → 1 ~1 → 0	~5 дає -6 ~0b101→ 0b110
i << j	<<=	Зсуває біти числа вліво на задану кількість	2 << 2 дасть 8. У двійковому вигляді 2

	i = i << j	позицій. Зсув вліво на N позицій еквівалентно множенню числа на 2^N. Таким чином, `43<<4 == 43*math.pow(2,4)`	це 0b10. Зсув вліво на 2 біти дає 0b1000, що в десятковій системі числення означає 8.
i >> j	>>= i = i >> j	Зсуває біти числа вправо на задане число позицій. Зсув числа вправо на N позицій еквівалентно діленню числа на 2^N .	11 >>1 дасть 5. У двійковій системі числення 11 становить 0b1011, що при зміщенні на 1 біт вправо, дає 0b101, а це, десяткове 5.

Приоритет побітних операторів наступний (в порядку спадання): ~; << >>; &; ^ |.

Наступні приклади демонструють виконання побітових операцій в Python.

```
>>> z = 7 & 5
>>> print(bin(7), bin(5), z, "{0:08b}".format(z))
0b111 0b101 5 00000101
>>> z = 7 | 5
>>> print(bin(7), bin(5), z, "{0:08b}".format(z))
0b111 0b101 7 00000111
>>> z = 7 ^ 5
>>> print(bin(7), bin(5), z, "{0:08b}".format(z))
0b111 0b101 2 00000010
>>> z = z ^ 5
>>> print(bin(z), bin(5), z, "{0:08b}".format(z))
0b111 0b101 7 00000111
>>> z = ~z
>>> print(bin(z), z, "{0:08b}".format(z))
-0b1000 -8 -0001000
```

Виконайте наступні вправи, які демонструють можливості побітових операторів.

Занулення останніх i біт числа A:

1. A>>=i
2. A<<=i

або

1. A &= (~0)<<i

Піднесення 2 до степеня n:

1. A = 1<<n;

Сума 2 в степені n та 2 в степені m:

1. `if (n==m)`
2. `  A = 1<<(n+1)`
3. `else`
4. `  A = 1<<n | 1<<m`

Встановлення i-го біту числа A рівним 1:

1. A |= (1<<i)

Інвертувати i-ий біт числа A:

1. A ^= (1<<i)

Встановити i-ий біт числа A рівним 0:

1. A &= ~(1<<i)

Занулити всі крім останніх i бітів числа A:

1. A &= (1<<i) - 1

Визначити значення i-ого біту числа A:

1. bitValue = (A>>i)&1;

Вивести значеня байта побітно:

1. `for i in range(8):`
2. `  print (n>>i & 1);`

Обнулити крайній правий одиничний біт числа A

1. `A &= A-1;`

2. Побітові операції дозволяють легко змінювати регістр літер.

```
chr(ord('A')|ord(' '))
'a'
chr(ord('a')&ord('_'))
'A'
```

How does above solutions work?

The trick lies in ASCII codes of 'A'-'Z' and 'a'-'z' –

'A' – 01000001 'a' – 01100001
'B' – 01000010 'b' – 01100010
'C' – 01000011 'c' – 01100011
'D' – 01000100 'd' – 01100100
'E' – 01000101 'e' – 01100101
and so on...

If we carefully analyze, we will notice that ASCII codes of lowercase and uppercase characters differ only in their third significant bit. For uppercase characters, the bit is 0 and for lowercase characters the bit is 1. If we could find a way to set/unset that particular bit, we can easily invert case of any character. Now space ' ' has ASCII code of `00100000` and '_' has ASCII code of `01011111`.

- If we take OR of an uppercase characters with ' ', the third significant bit will be set and we will get its lowercase equivalent.
- If we take AND of a lowercase character with '_', the third significant bit will be unset and we will get its uppercase equivalent.

3. Дослідіть як працює функція по перетворенню цілих чисел у двійковий вигляд (список бітів) та функція, яка робить зворотнє перетворення.

```
def int2bin(n):
    'From positive integer to list of binary bits, msb at index 0'
    if n:
        bits = []
        while n:
            n,remainder = divmod(n, 2)
            bits.insert(0, remainder)
        return bits
    else: return [0]

def bin2int(bits):
    'From binary bits, msb at index 0 to integer'
    i = 0
    for bit in bits:
        i = i * 2 + bit
    return i
```

Представлення чисел з плаваючою комою

Стандарт IEEE754 визначає формат представлення чисел з плаваючою комою. Числа з плаваючою комою представляються у форматі знак (s), мантиса (M) та порядок (E) наступним чином:

$(-1)^s \times 1.M \times 2^E$

Стандарт IEE754-2008 передбачає представлення чисел не тільки з основою 2, але і чисел з основою 10 - *десяткові* (decimal) числа с плаваючою комою.

В числах подвійної точності (float/double) порядок складється з 11 бітів, а мантиса – з 53 бітів. На стрінці за адресою http://www.binaryconvert.com можна дослідити представлення чисел з плаваючою комою у двійковому вигляді.

За допомогою модуля sys можна дізнатися як представлено float в Python

```
>>> sys.float_info
sys.float_info(max=1.7976931348623157e+308, max_exp=1024,
max_10_exp=308, min=2.2250738585072014e-308, min_exp=-1021,
min_10_exp=-307, dig=15, mant_dig=53, epsilon=2.220446049250313e-
16, radix=2, rounds=1)
```

Атрибут	Опис
float_info.epsilon	Різниця між 1 та найближчим float числом що більше за 1.0
float_info.dig	Кількість десяткових знаків, які можуть бути представлені без змін після округлення.
float_info.mant_dig	Кількість цифр мантиси в системі числення за основою, яка вказана в атрибуті float_info.radix.
float_info.max	Максимально можливе число з плаваючою комою.
float_info.max_exp	Максимальна експонента в системі числення за основою, яка вказана в атрибуті float_info.radix.
float_info.max_10_exp	Максимальна експонента в системі числення за основою 10.
float_info.min	Мінімально можливе додатнє число с плаваючою комою.
float_info.min_exp	Мінімальна експонента в системі числення за основою, яка вказана в атрибуті float_info.radix.
float_info.min_10_exp	Мінімальна експонента в системі числення за основою 10.
float_info.radix	Основа системи числення для показника степені.
float_info.rounds	Алгоритм округлення (-1 – не визначено, 0 – в сторону нуля, 1 – до найближчого значення, 2 – в сторону додатньої безкінечності, 3 – в сторону відємної безкінечності)

Модуль math містить декілька корисних функцій для роботи з float, які дозволяють

math.frexp(X) – обчислити мантису й експоненту числа.

math.ldexp(X, I) - $X * 2^i$. Функція, обернена до функції math.frexp().

math.modf(X) – визначається дробова та ціла частини числа X.

math.trunc(X) – відсікається дробова частина.

Представлення чисел за допомогою модулів decimal, fractions

Відомо, що використання типу float для представлення грошових сум або розрахунку відсотків банківського рахунку буде призводити до накопичення похибки. Для уникнення цих проблем потрібно використовувати модуль decimal, який дозволяє представити число як *знак*, *набір цифр* та *положення десяткової коми* — тобто число представлється без округления.

Для використання такого представлення чисел потрібно імпортувати модуль, а потім перетворити число:

```
>>> import decimal
>>> decimal.Decimal("4.31")
Decimal('4.31')
>>> decimal.Decimal("4.31") + decimal.Decimal("1.10")
Decimal('5.41')
```

Всі стандартні арифметичні операції з *decimal* працюють аналогічно до float та int.

За замовчуванням точність decimal 28 чисел в дробовій частині. Точність можна змінювати, наприклад для операцій з копійками необхідна точність 2 знаки:

```
>>> Decimal("1.10") / 3
Decimal('0.3666666666666666666666666667')

>>> decimal.getcontext().prec = 2
>>> Decimal('1.10') / 3
Decimal('0.37')
```

Decimal дозволяє налаштовувати не тільки точність, але і правила заокруглення та ще багато інших параметрів.

Для здійснення операцій зі звичайними дробами можна використовувати модуль fractions. Тип float не дозволяє провести точні обчислення

```
>>> 7/71*71 == 7
False
```

а за допомогою модуля *fractions* такі дії виконуються наступним чином:

```
>>> import fractions
>>> fractions.Fraction(7, 71) * 71 == 7
True
```

Література:

Ronald T. Kneusel Numbers and Computers Springer 2015.
http://www.techiedelight.com