Interning

Introduction

Interning is re-using objects of equal value on-demand instead of creating new objects. This is done for memory efficiency. Frequently used for numbers and strings in different programming languages.

```
a = 120
b = 120

print(a is b) # True

c = 2000
d = 2000

print(c is d) # False
```

In the above code, `120` is intered by the Python interpreter but not `2000`. Python's integer interning is done only for numbers in the range: `[-5, 256]`

Python interpreter also interns small strings.

```
a = "abcd"
b = "abcd"

print(a is b) # True

# Both text are the same
c = "Lorem ipsum dolor sit amet consectetur adipisicing elit. Consequuntur perferendis iste ipsa nat
d = "Lorem ipsum dolor sit amet consectetur adipisicing elit. Consequuntur perferendis iste ipsa nat

print(c is d) # False
```

Strings in python can be manually interned using `sys.intern` function.

```
a,b=8,8
c=8
```

```
d=8
```

Likewise, in the above code, only 1 integer object is created.

Practice Resources

Programs

The programs are listed in no specific order.

1. **is prime number**: A program that takes in a number n and outputs whether its a prime number or not.
2. **factors**: Take in a number from user. Output all of its factors.
3. **n-th factorial**: A program that takes in a number n and outputs n-th factorial.
4. **is perfect number**: A program that takes in a number n and outputs whether its a perfect number.
5. **fibonacci numbers**: A program that takes in a number n and prints all fibonacci numbers less than or equal to n.
6. **determinant of matrix**: Take in a matrix from user. Output the determinant of the matrix. First try for 2×2. Then go higher-ordered matrices.
7. **pascal's triangle**: Take n from user input. Print pascal's triangle to n rows.
8. **is valid palindrome**: Take a string input from user. Output if the input is palindrome or not. A phrase is a palindrome if, after converting all uppercase letters into lowercase letters and removing all non-alphanumeric characters, it reads the same forward and backward. Alphanumeric characters include letters and numbers. Try not to use `[::-1]`.
9. **armstrong numbers**: Take n from user input. Print all armstrong numbers (in base 10, of course) between 0 and n (inclusive).
10. **letter analysis**: Take a text input from user. Find how many times each letter is being used in that string. Use a `dictionary` to store the data. Output the final results. Try to read the text from a `.txt` file as well.
11. **word length analysis**: Take a string input from user. Print length of each word separated by a space. Try to include the summary using a `dictionary`.
12. **letter expanding**: A program that converts *b3j8k2* to *bbbjjjjjjjjkk*. The number can be 1 to 99.
13. **binary addition**: Take in 2 numbers in binary (as strings) and output the sum of both numbers. Try not to use `bin` function.

14. **big integer addition**: Given *a very large integer* represented as a list, where each `digits[i]` is the i^{th} digit of the integer. The digits are ordered from most significant to least significant in left-to-right order. Increment the large integer by one and return the resulting array of digits. Don't construct a `int` object.

Platforms

- Codewars - https://codewars.com (my most preferred one)
- HackerRank - https://hackerrank.com
- Leetcode - https://leetcode.com (my least preferred one)

> ⚠ **Hard Problems**
>
> If a problem from one of these platforms feels too hard for you, you can just skip and do another problem.

One's & Two's Complement

One's complement

The ones' complement of a binary number is the value obtained by flipping all the bits in the binary representation of the number.

- If one's complement of a is b, then one's complement of b is a.
- Binary representation of $a + b$ will include all 1 s.

One's complement system

In which negative numbers are represented by the inverse of the binary representations of their corresponding positive numbers. First bit denotes the sign of the number.

- Positive numbers are the denoted as basic binary numbers with 0 as the MSB.
- Negative values are denoted by the one's complement of their absolute value.

For example, to find the one's complement system representation of -7, one's complement of 7 must be found. $7 = 0111_2$. One's complement of -7 is 1000.

Two’s complement

In which negative numbers are represented using the MSB (sign bit).

If MSB is:

- 1 : negative
- 0 : positive

Positive numbers are represented as basic binary numbers with an additional 0 as the sign bit.

For example:

Following equation can be used to convert a number in two’s complement form to decimal.

$$b = -2^{n-1}b_{n-1} + \sum_{k=0}^{n-2} 2^k b_k$$

Steps

1. Starting with the absolute binary representation of the number
2. Add a leading 0 bit being a sign bit
3. Find the one’s complement: flip all bits (which effectively subtracts the value from -1)
4. Add 1, ignoring any overflows

Floating-point representation

IEEE 754 standard.

2 types:

- single precision
- double precision

Single precision

Uses 32 bits.

- sign bit - 1 bit
- exponent - 8 bit
- mantissa - 23 bit

Sign bit

0 if positive or zero. 1 if negative.

Exponent

Exponent field range - $[0, 255]$. In this range $[1, 254]$ is defined for normal numbers. 0 and 255 are reserved for subnormal, infinite, signed zeros and NaN.

To support negative exponents, we subtract 127 (half of 254) from this range. $[-126, 127]$. This range is the representable range.

Mantissa

In scientific notation, the part that doesn't contain the base and the power.

In binary scientific notation, there will always be exactly one 1 bit before the dot. So we don't include that one.

Example

Take 31.3125.

- In binary: 1111.0101_2
- In binary scientific notation: $1.1110101_2 \times 2^3$
- Add 127 to exponent: 130
- Convert exponent to binary 10000010
- Write the final result: $0\ 10000010\ 00000000000000001110101$

Take 0.125.

- In binary: -0.001_2
- In binary scientific notation: $-1.0_2 \times 2^{-3}$
- Add 127 to exponent: 124
- Convert exponent to binary 01111100
- Write the final result: $1\ 01111100\ 00000000000000000000000$

Double precision

Uses 64 bits.

- sign bit - 1 bit
- exponent - 11 bit
- mantissa - 53 bit

Sign bit

0 if positive or zero. 1 if negative.

Exponent

Exponent field range - $[0, 2047]$. In this range $[1, 2046]$ is defined for normal numbers. 0 and 2047 are reserved for subnormal, infinite, signed zeros and NaN.

To support negative exponents, we subtract 1023 (half of 2046) from this range. $[-1022, 1023]$. This range is the representable range.

Mantissa

In scientific notation, the part that doesn't contain the base and the power.

In binary scientific notation, there will always be exactly one 1 bit before the dot. So we don't include that one.

Example

Take 31.3125.

- In binary: 1111.0101_2
- In binary scientific notation: $1.1110101_2 \times 2^3$
- Add 1023 to exponent: 1026
- Convert exponent to binary: 10000000010
- Write the final result:
  $0\ 10000000010\ 0000000000000000000000000000000000000000000000001110101$

Take 0.125.

- In binary: -0.001_2
- In binary scientific notation: $-1.0_2 \times 2^-3$
- Add 1023 to exponent: 1020
- Convert exponent to binary: 1111111100
- Write the final result:
  $1\ 1111111100\ 0000000000000000000000000000000000000000000000000000$