Summary | Programming Fundamentals

Introduction

> **Note**
>
> Programming Fundamentals is probably the less-organized section at the moment. Let me know how I can improve this.

This module includes 3 sections:

- Programming basics (with python v3.10.9)
- Theories beyond coding
- Hardware

Important points

Confusion about unit prefixes

In computing, the prefix *kilo* —just like other prefixes— has been used to refer either 2^{10} or 10^3 depending on the context.

- 10^3 - Marketing of disk capacities (by disk manufacturers)
- 2^{10} - Memory capacities, and file sizes, disk capacities by operating systems

To avoid this confusion, 2 unit prefixes are used while measuring amounts of data.

- SI prefixes
  Defined by ISO. Based on powers of 10^3. Examples: kilo, mega, giga.
- Binary prefixes
  Defined by IEC. Based on powers of 2^{10}. Examples: kibi, mebi, gibi.

Interning

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.

15. **stack implementation**
16. **queue implementation**

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.

Number Systems

A writing system for expressing numbers. Each number system defines a set of symbols that each represent a specific value.

Base (or radix)

Number of symbols defined by a number system.

Commonly used number systems

- Base 10 - 0 - 9
- Base 2 - $0, 1$
- Base 8 - 0 - 7
- Base 16 - 0 - 9, A - F

> **Caution**
>
> These are required for s1:
>
> - Converting integers and floats between number systems
> - Addition, subtraction, multiplication, division in base 2
>
> But I don’t know how to include it in a easy-to-understand way. ⚫

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:
> $0\ 10000000010\ 00000000000000000000000000000000000000000000001110101$
>
> 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:
> $1\ 1111111100\ 00000000000000000000000000000000000000000000000000000$

String Representation

Software Engineering

Software

Refers to all the related things that are required to make a software system work.

Includes:

- programs
- configuration files
- system and user documentation
- user support system
- bug fixes and updates

Software engineering

An engineering discipline that is concerned with all aspects of software production. From the initial stage of writing the requirements to maintaining it while being used.

Software process

Set of activities that are associated with the development of a software product.

Fundamental activities that are common to all types of software development processes:

- Specification - defining the software to be produced and the runtime constraints
- Development - design and development of the software
- Validation - testing phase to check if the software meets the specifications
- Evolution - software is modified to adapt to new specifications

Waterfall

All before-mentioned activities are done sequentially, as clear separate phases. One phase is completed before the next phase is started.

Iterative & incremental

System is developed in iteration. Smaller parts of the system is completed in each iteration, that includes:

- Small amount of requirements specification
- Design and development for the specification
- Validation for the developed parts

Component based

Existing components are combined to implement the system. Main concentration is on the integration of the components.

Quality of software

Can be measured using these aspects:

- Maintainability - how easy it is to making changes
- Dependability - how secure, reliable it is to failures or other unusual activities
- Efficiency - how efficiently hardware resources (such as memory, processor time, disk space) are used
- Usability - how easy it is to use the software from user’s perspective
- Robustness - how resilient it is to invalid inputs

Challenges in software engineering

- Complexity
  - Essential - inherent, difficult to overcome
  - Accidental - not inherent, can be overcome
- Conformity
- Changeability - expected to be changeable to greater extent
- Invisibility - not visualizable
- Can’t guarantee defect free software - no amount of testing can prove absence of defects